ATTORNEY'S DOCKET NO.
TI-29958

APPLICATION FOR UNITED STATES PATENT

DECLARATION AND POWER OF ATTORNEY

As a below named inventor, I declare that my residence, post office address and citizenship are as stated below next to my name; that I verily believe that I am the original, first and sole inventor if only one name is listed below, or an original, first and joint inventor if plural inventors are named below, of the subject matter which is claimed and for which a patent is sought on the invention entitled as set forth below, which is described in the attached specification; that I have reviewed and understand the contents of the specification, including the claims, as amended by any amendment specifically referred to in the oath or declaration; that no application for patent or inventor's certificate on this invention has been filed by me or my legal representatives or assigns in any country foreign to the United States of America; and that I acknowledge my duty to disclose information which is material to the patentability of this application in accordance with Title 37, Code of Federal Regulations, section 1.56;

I further declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true, and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under section 1001 of Title 18 of the United States Code, and that such willful false statements may jeopardize the validity of the application or any patent issuing thereon.

TITLE OF INVENTION:

Apparatus and Method for Microcontroller Debugging

POWER OF ATTORNEY: I HEREBY APPOINT THE FOLLOWING ATTORNEYS TO PROSECUTE THIS APPLICATION AND TRANSACT ALL BUSINESS IN THE PATENT AND TRADEMARK OFFICE CONNECTED THEREWITH.

J. Dennis Moore, Reg. No. 28,885
Mark E. Courtney, Reg. No. 36,491
W. James Brady III, Reg. No. 32,080
Frederick J. Telecky, Jr., Reg. No. 29,979
William B. Kempler, Reg. No. 28,228
Jay M. Cantor, Reg. No. 19,906
Daniel Swayze, Reg. No. 34,478

SEND CORRESPONDENCE TO:		DIRECT TELEPHONE CALLS TO:
J. Dennis Moore Texas Instruments Incorporated PO Box 655474, M/S 3999 Dallas, TX 75265		J. Dennis Moore (972) 917-5646
NAME OF INVENTOR: (1) Brian Tse Deng	NAME OF INVENTOR: (2) Jian Yu	NAME OF INVENTOR: (3) Fengchun Duan
RESIDENCE (City and State Only) Richardson, Texas	RESIDENCE (City and State Only) Plano, Texas	RESIDENCE (City and State Only) Coppell, Texas
Post Office Address: 1117 Windsong Trail Richardson, Texas 75081	Post Office Address: 2512 Cimmaron Dr. Plano, Texas 75025	Post Office Address: 427 Forest Ridge Dr. Coppell, Texas 75019
Country of Citizenship: United States	Country of Citizenship: P. R. China	Country of Citizenship: Canada
Signature of Inventor: Brian Tse Deng	Signature of Inventor: Jian Yu	Signature of Inventor: Fengchun Duan
Date: 12/28/2000	Date: 12/28/2000	Date: 12/28/2000

APPENDIX


```
--------------------------------------------------------------------------------
--          debugger logic presentation
--------------------------------------------------------------------------------

signal bpsta_we_l       : std_logic; -- BD

signal bpl_we_l         : std_logic_vector( 4 downto 1); -- C7, C4, C1, BE
signal bph_we_l         : std_logic_vector( 4 downto 1); -- C8, C5, C2, BF
signal bnk_we_l         : std_logic_vector( 4 downto 1); -- C9, C6, C3, C0

signal jtml_we_l        : std_logic; -- CA
signal jtmh_we_l        : std_logic; -- CB

signal sbk_we_l         : std_logic; -- CE
signal bpcrl_we_l       : std_logic; -- CF

signal scratch_pad_we_l : std_logic_vector(15 downto 1); -- D1 -> DF

-- ------------------------------------------------------------------------
-- ESFR registers
--
-- BPSTA: Break Point Status register address: hBD
-- ------------------------------------------------------------------------

signal bpsta_sb_reg_l   : std_logic;  -- BPSTA reg SB (stack trap)
signal bpsta_b_reg_l    : std_logic_vector(4 downto 1);  -- BPSTA reg B4-B1

signal debugger_active  : std_logic;  -- break point condition occur, in debugger mode and it
                                      -- should temporarily disable all break point function
signal bpsta_resume     : boolean;    -- when write 80h to BPSTA, write-protect ESFR[BE-CF]

signal esfr_unprotect_l : std_logic;  -- enable write to ESFR[BE-CF]
signal pc_update_l      : std_logic;  -- enable the PCH/PCL update

-- ------------------------------------------------------------------------
-- Break point registers, (LB, HB,BANK) 1-4
-- ------------------------------------------------------------------------

signal bpl1             : std_logic_vector(7 downto 0);  -- Break point reg-1 (LB)
signal bpl2             : std_logic_vector(7 downto 0);  -- Break point reg-2 (LB)
signal bpl3             : std_logic_vector(7 downto 0);  -- Break point reg-3 (LB)
signal bpl4             : std_logic_vector(7 downto 0);  -- Break point reg-4 (LB)

signal bph1             : std_logic_vector(7 downto 0);  -- Break point reg-1 (HB)
signal bph2             : std_logic_vector(7 downto 0);  -- Break point reg-2 (HB)
signal bph3             : std_logic_vector(7 downto 0);  -- Break point reg-3 (HB)
signal bph4             : std_logic_vector(7 downto 0);  -- Break point reg-4 (HB)

signal bnk1_ben         : std_logic;                     -- Break point bank reg-1 (BEN)
signal bnk2_ben         : std_logic;                     -- Break point bank reg-2 (BEN)
signal bnk3_ben         : std_logic;                     -- Break point bank reg-3 (BEN)
signal bnk4_ben         : std_logic;                     -- Break point bank reg-4 (BEN)

signal bnk1_b           : std_logic_vector(3 downto 0);  -- Break point bank reg-1 (B[3:0])
signal bnk2_b           : std_logic_vector(3 downto 0);  -- Break point bank reg-2 (B[3:0])
signal bnk3_b           : std_logic_vector(3 downto 0);  -- Break point bank reg-3 (B[3:0])
signal bnk4_b           : std_logic_vector(3 downto 0);  -- Break point bank reg-4 (B[3:0])

signal bp1_adr          : std_logic_vector(15 downto 0); -- Break point 1 address
signal bp2_adr          : std_logic_vector(15 downto 0); -- Break point 2 address
signal bp3_adr          : std_logic_vector(15 downto 0); -- Break point 3 address
signal bp4_adr          : std_logic_vector(15 downto 0); -- Break point 4 address

-- ------------------------------------------------------------------------
-- Jump to Monitor address LB, HB, stack break point register adr: CA, CB, CE
-- ------------------------------------------------------------------------

signal jtml_l           : std_logic_vector(7 downto 0);
signal jtmh_l           : std_logic_vector(7 downto 0);
signal sbk_l            : std_logic_vector(7 downto 0);

-- ------------------------------------------------------------------------
-- Break point control register     adr: CF
--
-- bpcrl_be_reg_l -> BE4, BE3, BE2, BE1 address break point enable
-- bpcrl_s_reg_l  -> Stack trap condition
-- bpcrl_bpe_reg_l -> global debugger enable
-- ------------------------------------------------------------------------

signal bpcrl_bpe_reg_l  : std_logic;
signal bpcrl_s_reg_l    : std_logic_vector( 1 downto 0);
signal bpcrl_be_reg_l   : std_logic_vector( 4 downto 1);  -- for BE4, BE3, BE2, BE1

-- ------------------------------------------------------------------------
-- PC copy register (LB, HB)        adr: F9 - FA
-- ------------------------------------------------------------------------

signal pcl_l            : std_logic_vector(7 downto 0);
signal pch_l            : std_logic_vector(7 downto 0);

-- ------------------------------------------------------------------------
-- debugger circuitry signals for stack trap and address break point trap
-- ------------------------------------------------------------------------
```

```
signal stack_trap_1      : std_logic;  -- detect stack trap when global debugger is [redacted]bled
signal stack_trap_d1     : std_logic;  -- latched stack trap
signal stack_trap_d2     : std_logic;
signal stack_trap_d3     : std_logic;
signal stack_trap_d4     : std_logic;
signal stack_trap_d5     : std_logic;

signal stack_trap_jump_1: std_logic;

signal bp1_trap_1        : std_logic;
signal bp2_trap_1        : std_logic;
signal bp3_trap_1        : std_logic;
signal bp4_trap_1        : std_logic;

signal bp1_trap_active   : std_logic;  -- bp1_trap_1 and (not bp1_trap_toggle_1)
signal bp2_trap_active   : std_logic;
signal bp3_trap_active   : std_logic;
signal bp4_trap_active   : std_logic;

-- ------------------------------------------------------------------------------
-- When first detect address break point condition, bpx_toggle_1 is 0. After return from
-- Monitor program, the return address match break point address and bpx_toggle_1 is 1, so
-- it won't trigger address break point condition to avoid this infinitive loop.
-- ------------------------------------------ ---------------------------------

signal bp1_toggle_1      : std_logic;
signal bp2_toggle_1      : std_logic;
signal bp3_toggle_1      : std_logic;
signal bp4_toggle_1      : std_logic;

signal jump_ready        : std_logic;  -- ready to jump to monitor program
-- signal jump_1            : std_logic;
signal jump_adr_sel      : std_logic_vector(1 downto 0);  -- select 02H, jtmh_., jtml_1

signal jump_proga_en     : std_logic;  -- jump_proga_en <= prog_mux_control_1 and Proga_en_i
signal clear_jump_1      : boolean;
signal prog_mux_control_1: std_logic;  -- prog_mux_control_1 = 1, select jump address
                                       -- prog_mux_control_1 = 0, normal operation

-- ------------------------------------------------------------------------------
-- external SFR register block
-- ------------------------------------------------------------------------------

-- ------------------------------------------------------------------------------
-- BPSTA: Break Point Status Register  -- address hBD
--
-- B4~B3, B2, B1 -> break point address X trigger break condition
-- SB --> Stack Trap trigger break condition
--
-- When break condition occur, it will set one of B4, B3, B2, B1, SB.
--
-- When write h80 to this register it will clear reg. to 0, write-protect ESFR[BE-CF]
-- When write h55 to this register it will unprotect ESFR[BE-CF].
-- ------------------------------------------------------------------------------

bpsta_resume <= (bpsta_we_1 = '1' and Destin_do_i(7 downto 0) = "10000000");  -- write 80h to BPSTA

bpsta_reg: process (Clk_i, Resetz_exclude_USB_i)
begin
  if (Resetz_exclude_USB_i = '0') then  -- asynchronous reset
    bpsta_b_reg_1    <= "0000";
    bpsta_sb_reg_1   <= '0';

  elsif (Clk_i = '1' and Clk_i'event) then

    if (bpsta_resume) then  -- synchronous reset
      bpsta_b_reg_1    <= "0000";
      bpsta_sb_reg_1   <= '0';
    else
      if (bp1_trap_active = '1') then
        bpsta_b_reg_1(1)    <= '1';
      end if;

      if (bp2_trap_active = '1') then
        bpsta_b_reg_1(2)    <= '1';
      end if;

      if (bp3_trap_active = '1') then
        bpsta_b_reg_1(3)    <= '1';
      end if;

      if (bp4_trap_active = '1') then
        bpsta_b_reg_1(4)    <= '1';
      end if;

      if (stack_trap_jump_1 = '1') then
        bpsta_sb_reg_1    <= '1';
      end if;

    end if;
  end if;
end process bpsta_reg;
```

```
  ------------------------------------------------------------------------------
  debugger_active -> break point condition occurs, it should temporarily disable break point
                     trigger function.
  ------------------------------------------------------------------------------

bugger_active <= bpsta_b_reg_1(1) or bpsta_b_reg_1(2) or bpsta_b_reg_1(3) or bpsta_b_reg_1(4) or
                 bpsta_sb_reg_1;

  ---------------------------------------------------------------
  reg_unprotect_1 = 0 -> ESFR[BE-CF] are write protected.
  reg_unprotect_1 = 1 -> ESFR[BE-CF] are unprotected.

  When jump to Monitor, automatically unprotect.
  ---------------------------------------------------------------

otect_reg: process (Clk_i, Resetz_exclude_USB_i)
gin
  if (Resetz_exclude_USB_i = '0') then
     reg_unprotect_1  <=  '0'; -- reset to write-protected after power on

  elsif (Clk_i = '1' and Clk_i'event) then
     if ( (bpsta_we_1 = '1' and Destin_do_i(7 downto 0) = "01010101") or -- write 55H
          (prog_mux_control_1 = '1')  -- Jump to Monitor
        )then
        reg_unprotect_1  <=  '1'; -- write unprotected when write 55H
     elsif (bpsta_resume) then
        reg_unprotect_1  <=  '0'; -- write-protected when write 80H
     end if;
  end if;
d process protect_reg;

  ---------------------------------------------------------------
  Write h80 to BPSTA: Break Point Status register to enable
  PCL/PCH update.

  When jump is ready, it disable PCL/PCH update.
  ---------------------------------------------------------------

:_update_reg: process (Clk_i, Resetz_exclude_USB_i)
:gin
  if (Resetz_exclude_USB_i = '0') then
     pc_update_1  <=  '0';

  elsif (Clk_i = '1' and Clk_i'event) then
     if (bpsta_resume) then
        pc_update_1  <=  '1'; -- Write 80H to BPSTA enable PCH/PCL update
     elsif (jump_ready = '1' ) then  -- break point address is stored in PCL/PCH
        pc_update_1  <=  '0'; -- disable the update when reach breakpoint
     end if;
  end if;

id process pc_update_reg;

  ---------------------------------------------------------------
  Break point register-1 (LB)  address - hex BE
  ---------------------------------------------------------------

)ll_reg: process (Clk_i, Resetz_exclude_USB_i)
:gin
 if (Resetz_exclude_USB_i = '0') then
   bpl1    <= "00000000";  -- power-up default value
 elsif (Clk_i = '1' and Clk_i'event) then

   if ((bpl_we_1(1) = '1') and (reg_unprotect_1 = '1')) then
     bpl1 <=  Destin_do_i(7 downto 0);
   end if;
 end if;
id process bpl1_reg;

  ---------------------------------------------------------------
  Break point register-1 (HB)  address - hex BF
  ---------------------------------------------------------------

)hl_reg: process (Clk_i, Resetz_exclude_USB_i)
:gin
 if (Resetz_exclude_USB_i = '0') then
   bph1    <= "00000000";  -- power-up default value
 elsif (Clk_i = '1' and Clk_i'event) then

   if ((bph_we_1(1) = '1') and (reg_unprotect_1 = '1')) then
     bph1 <=  Destin_do_i(7 downto 0);
   end if;
 end if;
id process bph1_reg;

  ---------------------------------------------------------------
  Break point bank register-1  address - hex C0
  ---------------------------------------------------------------

nk1_reg: process (Clk_i, Resetz_exclude_USB_i)
:gin
 if (Resetz_exclude_USB_i = '0') then
   bnk1_b   <= "0000";      -- power-up default value
   bnk1_ben <= '0';
```

```
  elsif (Clk_i = '1' and Clk_i'event) then

    if ((bnk_we_1(1) = '1') and (reg_unprotect_1 = '1')) then
      bnk1_b   <=  Destin_do_i(3 downto 0);  -- break point address - bank value
      bnk1_ben <=  Destin_do_i(7);  -- bank value comparison enable
    end if;
  end if;
end process bnk1_reg;

-- ---------------------------------------------------------------
-- Break point register-2 (LB)  address - hex C1
-- ---------------------------------------------------------------

bpl2_reg: process (Clk_i, Resetz_exclude_USB_i)
begin
  if (Resetz_exclude_USB_i = '0') then
    bpl2    <= "00000000";  -- power-up default value
  elsif (Clk_i = '1' and Clk_i'event) then

    if ((bpl_we_1(2) = '1') and (reg_unprotect_1 = '1')) then
      bpl2 <=  Destin_do_i(7 downto 0);
    end if;
  end if;
end process bpl2_reg;

-- ---------------------------------------------------------------
-- Break point register-2 (HB)  address - hex C2
-- ---------------------------------------------------------------

bph2_reg: process (Clk_i, Resetz_exclude_USB_i)
begin
  if (Resetz_exclude_USB_i = '0') then
    bph2    <= "00000000";  -- power-up default value
  elsif (Clk_i = '1' and Clk_i'event) then

    if ((bph_we_1(2) = '1') and (reg_unprotect_1 = '1')) then
      bph2 <=  Destin_do_i(7 downto 0);
    end if;
  end if;
end process bph2_reg;

-- ---------------------------------------------------------------
-- Break point bank register-2  address - hex C3
-- ---------------------------------------------------------------

bnk2_reg: process (Clk_i, Resetz_exclude_USB_i)
begin
  if (Resetz_exclude_USB_i = '0') then
    bnk2_b   <= "0000";      -- power-up default value
    bnk2_ben <= '0';
  elsif (Clk_i = '1' and Clk_i'event) then

    if ((bnk_we_1(2) = '1') and (reg_unprotect_1 = '1')) then
      bnk2_b   <=  Destin_do_i(3 downto 0);  -- break point address - bank value
      bnk2_ben <=  Destin_do_i(7);  -- bank value comparison enable
    end if;
  end if;
end process bnk2_reg;

-- ---------------------------------------------------------------
-- Break point register-3 (LB)  address - hex C4
-- ---------------------------------------------------------------

bpl3_reg: process (Clk_i, Resetz_exclude_USB_i)
begin
  if (Resetz_exclude_USB_i = '0') then
    bpl3    <= "00000000";  -- power-up default value
  elsif (Clk_i = '1' and Clk_i'event) then

    if ((bpl_we_1(3) = '1') and (reg_unprotect_1 = '1')) then
      bpl3 <=  Destin_do_i(7 downto 0);
    end if;
  end if;
end process bpl3_reg;

-- ---------------------------------------------------------------
-- Break point register-3 (HB)  address - hex C5
-- ---------------------------------------------------------------

bph3_reg: process (Clk_i, Resetz_exclude_USB_i)
begin
  if (Resetz_exclude_USB_i = '0') then
    bph3    <= "00000000";  -- power-up default value
  elsif (Clk_i = '1' and Clk_i'event) then

    if ((bph_we_1(3) = '1') and (reg_unprotect_1 = '1')) then
      bph3 <=  Destin_do_i(7 downto 0);
    end if;
  end if;
end process bph3_reg;

-- ---------------------------------------------------------------
-- Break point bank register-3  address - hex C6
-- ---------------------------------------------------------------
```

```
bnk3_reg: process (Clk_i, Resetz_exclude_USB_i)
begin
  if (Resetz_exclude_USB_i = '0') t
    bnk3_b   <= "0000";     -- powe  default value
    bnk3_ben <= '0';
  elsif (Clk_i = '1' and Clk_i'event) then

    if ((bnk_we_l(3) = '1') and (reg_unprotect_l = '1')) then
      bnk3_b   <=  Destin_do_i(3 downto 0);  -- break point address - bank value
      bnk3_ben <=  Destin_do_i(7);  -- bank value comparison enable
    end if;
  end if;
end process bnk3_reg;

-- ----------------------------------------------------------------------
-- Break point register-4 (LB)  address - hex C7
-- ----------------------------------------------------------------------

bpl4_reg: process (Clk_i, Resetz_exclude_USB_i)
begin
  if (Resetz_exclude_USB_i = '0') then
    bpl4    <= "00000000";  -- power-up default value
  elsif (Clk_i = '1' and Clk_i'event) then

    if ((bpl_we_l(4) = '1') and (reg_unprotect_l = '1')) then
      bpl4 <=  Destin_do_i(7 downto 0);
    end if;
  end if;
end process bpl4_reg;

-- ----------------------------------------------------------------------
-- Break point register-4 (HB)  address - hex C8
-- ----------------------------------------------------------------------

bph4_reg: process (Clk_i, Resetz_exclude_USB_i)
begin
  if (Resetz_exclude_USB_i = '0') then
    bph4    <= "00000000";  -- power-up default value
  elsif (Clk_i = '1' and Clk_i'event) then

    if ((bph_we_l(4) = '1') and (reg_unprotect_l = '1')) then
      bph4 <=  Destin_do_i(7 downto 0);
    end if;
  end if;
end process bph4_reg;

-- ----------------------------------------------------------------------
-- Break point bank register-4  address - hex C9
-- ----------------------------------------------------------------------

bnk4_reg: process (Clk_i, Resetz_exclude_USB_i)
begin
  if (Resetz_exclude_USB_i = '0') then
    bnk4_b   <= "0000";     -- power-up default value
    bnk4_ben <= '0';
  elsif (Clk_i = '1' and Clk_i'event) then

    if ((bnk_we_l(4) = '1') and (reg_unprotect_l = '1')) then
      bnk4_b   <=  Destin_do_i(3 downto 0);  -- break point address - bank value
      bnk4_ben <=  Destin_do_i(7);  -- bank value comparison enable
    end if;
  end if;
end process bnk4_reg;

-- ----------------------------------------------------------------------
-- Jump to Monitor adr reg (LB) address - hex CA
-- ----------------------------------------------------------------------

jtml_regs: process (Clk_i, Resetz_exclude_USB_i)
begin
  if (Resetz_exclude_USB_i = '0') then
    jtml_l    <= "00000000";

  elsif (Clk_i = '1' and Clk_i'event) then

    if (jtml_we_l = '1' and reg_unprotect_l = '1') then
      jtml_l <=  Destin_do_i(7 downto 0);
    end if;
  end if;
end process jtml_regs;

-- ----------------------------------------------------------------------
-- Jump to Monitor adr reg (HB) address - hex CB
-- ----------------------------------------------------------------------

jtmh_regs: process (Clk_i, Resetz_exclude_USB_i)
begin
  if (Resetz_exclude_USB_i = '0') then
    jtmh_l    <= "00000000";

  elsif (Clk_i = '1' and Clk_i'event) then

    if (jtmh_we_l = '1' and reg_unprotect_l = '1') then
```

```
      jtmh_1 <=  Destin_do_i(7 downto
    end if;
  end if;
end process jtmh_regs;

-- ---------------------------------------------------------------
-- Stack break point register   address - hex CE
-- ---------------------------------------------------------------

sbk_regs: process (Clk_i, Resetz_exclude_USB_i)
begin
  if (Resetz_exclude_USB_i = '0') then
    sbk_1    <= "00000000";

  elsif (Clk_i = '1' and Clk_i'event) then

    if (sbk_we_1 = '1' and reg_unprotect_1 = '1') then
      sbk_1 <=  Destin_do_i(7 downto 0);
    end if;
  end if;
end process sbk_regs;

-- ---------------------------------------------------------------
-- Break point control register address - hex CF
-- ---------------------------------------------------------------

bpcrl_reg: process (Clk_i, Resetz_exclude_USB_i)
begin
  if (Resetz_exclude_USB_i = '0') then
    bpcrl_bpe_reg_1  <=  '0';
    bpcrl_s_reg_1    <=  "00";
    bpcrl_be_reg_1   <=  "0000";

  elsif (Clk_i = '1' and Clk_i'event) then

    if (bpcrl_we_1 = '1' and reg_unprotect_1 = '1') then
      bpcrl_bpe_reg_1  <=  Destin_do_i(7);
      bpcrl_s_reg_1    <=  Destin_do_i(5 downto 4);
      bpcrl_be_reg_1   <=  Destin_do_i(3 downto 0);
    end if;
  end if;
end process bpcrl_reg;

-- ---------------------------------------------------------------
-- PC copy register (LB, HB)        address - hex F9, FA
-- ---------------------------------------------------------------

pc_reg: process (Clk_i, Resetz_exclude_USB_i)
begin
  if (Resetz_exclude_USB_i = '0') then
    pcl_1   <= (others => '0') ;
    pch_1   <= (others => '0');

  elsif (Clk_i = '1' and Clk_i'event) then

    if (pc_update_1 = '1' and Proga_en_i = '1') then  -- when not Monitor mode, update PCL
      pcl_1   <= Proga_i(7 downto 0);
      pch_1   <= Proga_i(15 downto 8);
    end if;
  end if;
end process pc_reg;

-- ---------------------------------------------------------------------------
-- ---------------------------------------------------------------------------
--
--  MCU Debugger Logic
--
-- ---------------------------------------------------------------------------
-- ---------------------------------------------------------------------------

-- When debugger is not active, global debugger enable and write to stack pointer
-- check whether break point condition is matched
-- directly use the Write_stack_ptr_i and Stack_pointer_i from M8051 to
-- check if the trap condition is met or not

stack_breakpoint: process (debugger_active, bpcrl_bpe_reg_1, Write_stack_ptr_i,
                           Stack_pointer_i, sbk_1, bpcrl_s_reg_1)
begin
  if (((not debugger_active) and bpcrl_bpe_reg_1 and Write_stack_ptr_i) = '1' ) then
    if ((Stack_pointer_i > sbk_1 and bpcrl_s_reg_1 = "11") or
        (Stack_pointer_i = sbk_1 and bpcrl_s_reg_1 = "01") or
        (Stack_pointer_i < sbk_1 and bpcrl_s_reg_1 = "10")) then

      stack_trap_1 <= '1';
    else
      stack_trap_1 <= '0';
    end if;
  else
    stack_trap_1 <= '0';
  end if;
end process stack_breakpoint;

-- This logic remembers the trap codition, and it can wait for
-- the last cycle of the current instruction
```

```
stack_regs: process (Clk_i, Resetz_exclude_USB_i)
begin
  if (Resetz_exclude_USB_i = '0') th
    stack_trap_d1    <= '0';

  elsif (Clk_i = '1' and Clk_i'event) then
    if (stack_trap_1 = '1') then
      stack_trap_d1  <=  '1';
    elsif (prog_mux_control_1 = '1') then  -- when in jump to monitor mode, clear stack trap latched signals
      stack_trap_d1  <=  '0';
    end if;
  end if;
end process stack_regs;

stack_trap_d2 <= stack_trap_1 or stack_trap_d1;  -- when stack_trap_1 is set, it may be the last cycle

-- ---------------------------------------------------------------------------------------------
-- trigger jump when last instruction cycle is reached, and set_stack_trap = 1
-- ---------------------------------------------------------------------------------------------

stack_trap_d3 <= Last_cyc_i and State_0_i and (not Internal_wait_i) and stack_trap_d2;

process (Clk_i, Resetz_exclude_USB_i)
begin
  if (Resetz_exclude_USB_i = '0') then
    stack_trap_d4 <= '0';
  elsif (Clk_i = '1' and Clk_i'event) then
    if (stack_trap_d3 = '1') then  -- extend stack trap time until jump to monitor
      stack_trap_d4 <= '1';
    elsif (prog_mux_control_1 = '1') then  -- when in jump to monitor mode, clear stack trap latched signals
      stack_trap_d4 <= '0';  -- clear when debugger is active
    end if;
  end if;
end process;

stack_trap_d5 <= stack_trap_d3 or stack_trap_d4;  -- hold stack trap condition until jump_1 = 1

-- ---------------------------------------------------------------------------------------------
-- When stack trap condition occurs, wait for program memory fetch cycle to jump to monitor
-- program.
-- ---------------------------------------------------------------------------------------------

stack_trap_jump_1 <= Proga_en_i and stack_trap_d5;

-- ---------------------------------------------------------------------------------------------
-- address break point logic
--
-- When detect stack trap, it will disable address break point function.
-- ---------------------------------------------------------------------------------------------

bp1_adr <= bph1 & bpl1;  -- break point 1 address - 16 bits

bp2_adr <= bph2 & bpl2;  -- break point 2 address - 16 bits

bp3_adr <= bph3 & bpl3;  -- break point 3 address - 16 bits

bp4_adr <= bph4 & bpl4;  -- break point 4 address - 16 bits

breakpoint_logic: process (bank_1, Proga_i, Proga_en_i, bpcrl_bpe_reg_1, stack_trap_d2, debugger_active,
                           bpcrl_be_reg_1, bnk1_ben, bnk2_ben, bnk3_ben, bnk4_ben, bnk1_b, bnk2_b,
                           bnk3_b, bnk4_b, bp1_adr, bp2_adr, bp3_adr, bp4_adr)
begin
  bp1_trap_1 <= '0';
  bp2_trap_1 <= '0';
  bp3_trap_1 <= '0';
  bp4_trap_1 <= '0';

  -- when blobal debugger enable and not stack trap and not debugger_active and program memory fetch cycle

  if ((bpcrl_bpe_reg_1 and (not (stack_trap_d2 or debugger_active)) and Proga_en_i) = '1') then

    if (bpcrl_be_reg_1(1) = '1' and (bnk1_b = bank_1 or bnk1_ben = '0') and Proga_i = bp1_adr) then
      bp1_trap_1 <= '1';

    elsif (bpcrl_be_reg_1(2) = '1' and (bnk2_b = bank_1 or bnk2_ben = '0') and Proga_i = bp2_adr) then
      bp2_trap_1 <= '1';

    elsif (bpcrl_be_reg_1(3) = '1' and (bnk3_b = bank_1 or bnk3_ben = '0') and Proga_i = bp3_adr) then
      bp3_trap_1 <= '1';

    elsif (bpcrl_be_reg_1(4) = '1' and (bnk4_b = bank_1 or bnk4_ben = '0') and Proga_i = bp4_adr) then
      bp4_trap_1 <= '1';

    end if;
  end if;
end process breakpoint_logic;

-- ---------------------------------------------------------------------------------------------
-- bpx_trap_active -> address break point condition is active, and will cause jump tom monitor
-- program.
--
-- bpx_trap_active is active when break point address is matching and it occur on odd number.
-- address break point occurs on even number when monitor return to break point address and it
```

```
-- should not trigger long jump to monitor program.
-- ------------------------------------------------------------------------------

bp1_trap_active <= bp1_trap_l and (not bp1_toggle_l);

bp2_trap_active <= bp2_trap_l and (not bp2_toggle_l);

bp3_trap_active <= bp3_trap_l and (not bp3_toggle_l);

bp4_trap_active <= bp4_trap_l and (not bp4_toggle_l);

jump_ready <= stack_trap_jump_l or bp1_trap_active or bp2_trap_active or bp3_trap_active or bp4_trap_active;

jump_gen: process (Clk_i, Resetz_exclude_USB_i)
begin
  if (Resetz_exclude_USB_i = '0') then
    prog_mux_control_l  <=  '0';

  elsif (Clk_i = '1' and Clk_i'event) then

    if (clear_jump_l) then  -- clear_jump_l is boolean
      prog_mux_control_l  <=  '0';
    elsif (jump_ready = '1') then
      prog_mux_control_l  <=  '1';
    end if;
  end if;
end process jump_gen;

jump_proga_en <= prog_mux_control_l and Proga_en_i;  -- jump to monitor and program memory fetch enable

process (Clk_i, Resetz_exclude_USB_i)
begin
  if (Resetz_exclude_USB_i = '0') then
    jump_adr_sel <= "00";

  elsif (Clk_i = '1' and Clk_i'event) then

    if (jump_proga_en = '1') then  -- when jump and fetch instruction
      if (clear_jump_l) then
        jump_adr_sel <= "00";  -- when clear jump, clear jump address select
      else
        jump_adr_sel <= (jump_adr_sel + 1);  -- other increment jump address select by 1
      end if;
    end if;
  end if;
end process;

jump_logic: process (jump_adr_sel, jtmh_l, jtml_l)
begin
  case jump_adr_sel is
    when "00"   => Jump_progdi_o  <= "00000010"; --02H, M8051 long jump instruction
    when "01"   => Jump_progdi_o  <= jtmh_l;     -- jump to monitor address reg. (HB)
    when "10"   => Jump_progdi_o  <= jtml_l;     -- jump to monitor address reg. (LB)
    when others => Jump_progdi_o  <= jtml_l;
  end case;
end process jump_logic;

-- ------------------------------------------------------------------------------
-- When in jump to monitor mode and program memory fetch and jump address select = 10, clear jump
-- to monitor mode (clear prog_mux_control_l and stack trap latched signals
-- ------------------------------------------------------------------------------

clear_jump_l <= (jump_proga_en = '1' and jump_adr_sel = "10");

toggle_reg: process (Clk_i, Resetz_exclude_USB_i)
begin
  if (Resetz_exclude_USB_i = '0') then
    bp1_toggle_l   <=  '0';
    bp2_toggle_l   <=  '0';
    bp3_toggle_l   <=  '0';
    bp4_toggle_l   <=  '0';

  elsif (Clk_i = '1' and Clk_i'event) then

    if (bp1_trap_l = '1') then
      bp1_toggle_l    <=  not bp1_toggle_l;
    end if;

    if (bp2_trap_l = '1') then
      bp2_toggle_l    <=  not bp2_toggle_l;
    end if;

    if (bp3_trap_l = '1') then
      bp3_toggle_l    <=  not bp3_toggle_l;
    end if;

    if (bp4_trap_l = '1') then
      bp4_toggle_l    <=  not bp4_toggle_l;
    end if;
  end if;
end process toggle_reg;
```